IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellants:	Gerard Chauvel, et al.	§	Confirmation No.:	2081
		§		
		§	Group Art Unit:	2181
Serial No.:	10/632,222	§		
		§	Examiner:	J. R. Moll
Filed:	July 31, 2003	§		
		§	Atty. Docket No.:	TI-35452 (1962-05416)
For:	Processor With Pre-Decode Logic That Detects A Prefix In An Instruction And Causes A Program Counter To Be Skipped	§		

APPEAL BRIEF

Mail Stop Appeal Brief – Patents Date: February 8, 2008
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

Appellants hereby submit this Appeal Brief in connection with the above-identified application. A Notice of Appeal and a Pre-Appeal Brief Request for Review were filed on December 8, 2006.

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST

The real party in interest is Texas Instruments, Inc. ("TI"), having headquarters in Dallas, Texas. TI is the assignee of record.

II. RELATED APPEALS AND INTERFERENCES

Appellants are unaware of any related appeals or interferences.

III. STATUS OF CLAIMS

Originally filed claims:	1-34.
Canceled claims:	32-34.
Presently rejected claims:	1-31.
Presently appealed claims:	1-31.

IV. STATUS OF AMENDMENTS

On February 7, 2008, Appellants filed an Amendment in accordance with 37 C.F.R. § 41.33(a) to correct antecedent basis issues. This Brief assumes entry of that Amendment.

V. SUMMARY OF CLAIMED SUBJECT MATTER

Various embodiments of the invention are described below. The scope of disclosure is not limited by the descriptions of the embodiments that follow. Citations to the specification have been provided to demonstrate where support may be found in the specification for various parts of the invention. Additional support may be found elsewhere in the application.

The claims are generally directed to systems and techniques for improving efficiency when decoding executable instructions. When one instruction is being decoded by a processor, the processor "pre-decodes" the next instruction. When pre-decoding the next instruction, the processor determines whether that next instruction contains a certain prefix. If so, the processor discards that prefix and changes its decoding behavior when decoding that next instruction. <u>The processor discards the prefix by adjusting a program counter</u>. The fact that the processor discards the prefix by adjusting the program counter (instead of adjusting hardware using signals) is significant at least because it confers many useful benefits. For example, adjusting software instead of hardware entails fewer points of potential failure, increased speed, increased efficiency, decreased space (real estate) requirements, *etc.*

Claim 1 is directed to a processor 102 that comprises instruction storage 130 in which instructions are stored. P. 9, l. 11; Figure 2. The processor 102 also comprises fetch logic 154 coupled to the instruction storage 130; the fetch logic 154 fetches instructions from the instruction storage 130. *Id.* The processor 102 further comprises decode logic 152 coupled to the fetch logic 154. Figure 2. The decode logic 152 decodes instructions fetched by the fetch logic 154. P. 9, ll. 22-23; Figure 2. The processor 102 still further comprises pre-decode logic 158 that is associated with the decode logic 152. *Id.* At least some of the instructions comprise a prefix. P. 21, l. 9. In parallel with the decode logic 152 decoding a current instruction, the pre-decode logic 158 determines whether a subsequent instruction comprises a prefix. P. 13, ll. 1-4; Figure 2. If so, the decode logic 152 causes a program counter 160 to skip the prefix, thereby precluding the decode logic 152 from receiving the prefix. P. 14, ll. 9-14; Figure 2. The behavior of the decode logic 152 is changed during decoding of the subsequent instruction. P. 14, ll. 7-8; Figure 2.

Claim 9 is directed to a method of decoding variable length instructions. The method comprises decoding a current instruction according to a first behavior and, while decoding the current instruction, pre-decoding a subsequent instruction to determine if the subsequent instruction includes a predetermined prefix. P. 13, ll. 1-4; Figure 2. The method further comprises, if the subsequent instruction includes the predetermined prefix, causing a program counter 160 to skip the predetermined prefix to thereby preclude decode logic 152 from receiving the prefix and changing the decoding of the subsequent instruction according to a second behavior. P. 14, ll. 9-14; Figure 2 and p. 14, ll. 7-8; Figure 2.

Claim 17 is directed to a system 100 that comprises a main processor unit 104 and a co-processor unit 102 coupled to the main processor unit 104. Figure 1. The co-processor unit 102 comprises decode logic 152 and pre-decode logic 158 associated with the decode logic 152. Figure 2. The decode logic 152 decodes a current instruction concurrently with the pre-decode logic 158 determining if a subsequent instruction comprises a prefix. P. 13, ll. 1-4; Figure 2. If the subsequent instruction comprises a prefix, a program counter 160 skips the prefix, thereby precluding the decode logic 152 from receiving the prefix. P. 14, ll. 9-14; Figure 2. The operation of the decode logic 152 operation changes during the decoding of the subsequent instruction. P. 14, ll. 7-8; Figure 2.

Dependent claim 21 is directed to the system 100 of claim 17, wherein the prefix value is equal to a Java wide instruction. P. 14, l. 18.

Dependent claim 22 is directed to the system 100 of claim 17, wherein the prefix value is equal to a Java impdep prefix. P. 14, l. 18.

Claim 25 is directed to a programmable device 102 that comprises a register 140 storing a location of a current instruction. Figure 2. The device 102 also comprises decode logic 152 and pre-decode logic 158 coupled to the decode logic 152. *Id.* In parallel, the decode logic 152 decodes the current instruction and the pre-decode logic

158 determines if a subsequent instruction includes a prefix. P. 13, ll. 1-4; Figure 2. If the subsequent instruction comprises the prefix, the register 140 skips the prefix of the subsequent instruction, thereby precluding the decode logic 152 from receiving the prefix. P. 14, ll. 9-14; Figure 2. The behavior of the decode logic 152 is changed for decoding of the subsequent instruction. P. 14, ll. 7-8; Figure 2.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Whether under 35 U.S.C. § 103(a) claims 1, 2, 7-12, 25 and 28-31 are rendered obvious in view of Chennupaty (US Pat. No. 6,014,735) and Narayan (US Pat. No. 6,161,172).

Whether under 35 U.S.C. § 103(a) claims 3-4, 13, 15-16 and 27 are rendered obvious in view of Chennupaty, Narayan and Google (*New Bytecodes For "Real" Java?).*

Whether under 35 U.S.C. § 103(a) claims 5-6, 14 and 26 are rendered obvious in view of Chennupaty, Narayan and JVM (The JavaTM Virtual Machine Specification).

Whether under 35 U.S.C. § 103(a) claims 17-20, 23 and 24 are rendered obvious in view of Chennupaty, Narayan and Nazomi (*Nazomi Introduces First Universal Java Accelerator Chip for Mobile Wireless Applications*).

Whether under 35 U.S.C. § 103(a) claim 21 is rendered obvious in view of Chennupaty, Narayan, Nazomi and Google.

Whether under 35 U.S.C. § 103(a) claim 22 is rendered obvious in view of Chennupaty, Narayan, Nazomi and JVM.

VII. ARGUMENT

A. Summary of Chennupaty

Chennupaty teaches a method and apparatus for encoding an instruction in an instruction set which uses a prefix code to qualify an existing opcode of an instruction. An opcode and an escape code are selected. The escape code is selected such that it is different from the prefix code and the existing opcode. The opcode, the escape code and the prefix code are combined to generate an instruction code which uniquely represents the operation performed by the instruction.

B. Rejections under 35 U.S.C. § 103(a)

1. Rejections in View of Chennupaty and Narayan

Claims 1, 2, 7-12, 25 and 28-31 stand rejected under 35 U.S.C. § 103(a) as allegedly obvious in view of Chennupaty and Narayan. Appellants elect claim 1 to be representative of this grouping of claims. The grouping should not be construed to mean the patentability of any of the claims may be determined in later actions (*e.g.*, actions before a court) based on the groupings. Rather, the presumption of 35 USC § 282 shall apply to each of these claims individually.

Claim 1 requires that "...the pre-decode logic determines whether a subsequent instruction comprises a prefix, in which case the decode logic causes a program counter to skip the prefix thereby precluding the decode logic from receiving the prefix..." The combination of Chennupaty and Narayan fails to teach or suggest precluding the decode logic from receiving the prefix by causing a program counter to skip the prefix.

Figure 4 of Chennupaty is reproduced below for convenience:

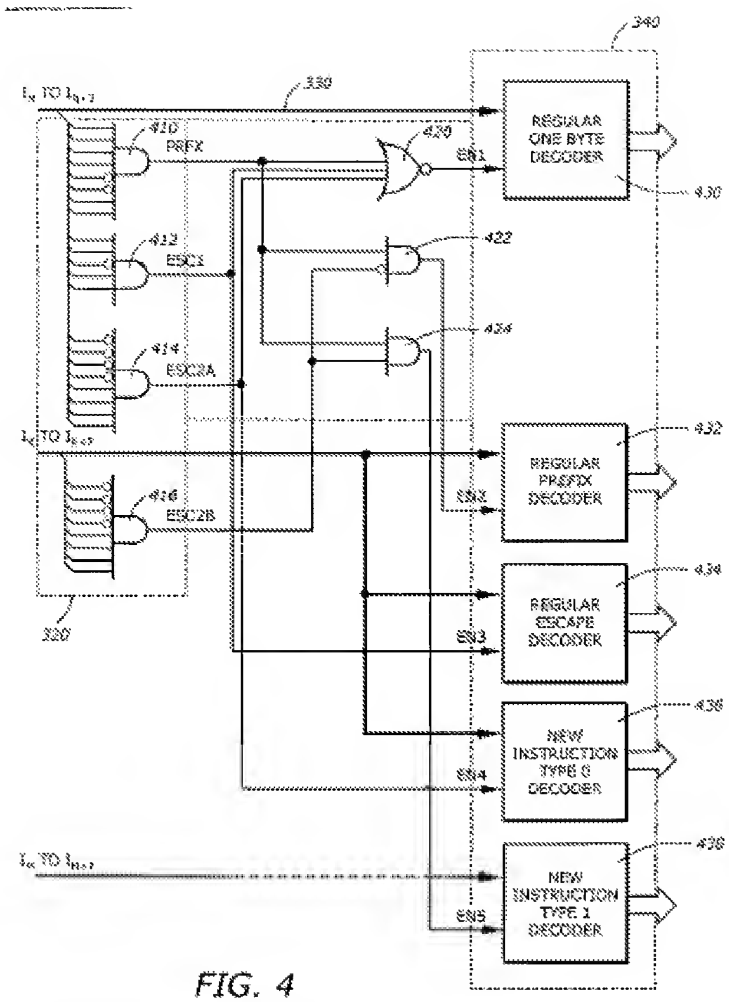

FIG. 4

Fig. 4 shows a decode unit containing a prefix and escape detector 320, a decoder enable circuit 330 and an opcode decoder 340. As described in col. 5, ll. 33-39, multiple bytes are provided to the decoder 340. Specifically, bytes I_N to I_{N+7}, I_K to I_{K+7} and I_M to I_{M+7} are provided to the decoder 340. In many cases, the byte I_N to I_{N+7} is a prefix. If the byte I_N to I_{N+7} is a prefix, col. 5, ll. 59-65 state that the output enable signal EN1 of NOR gate 420 is negated. Because the enable signal EN1 is negated, the decoder 430 is disabled, thereby precluding the decoder 430 (and the decoder 340 in general) from receiving the byte I_N to I_{N+7} (that is, the prefix).

Although the prefix is precluded from being received by the decoder 340, Appellants point out that the technique ***by which*** the prefix is kept out of the decoder 340 is different than what is required by claim 1. Claim 1 requires that the decode logic is precluded "from receiving the prefix" because "the decode logic causes a program counter to skip the prefix." That is not the case with Chennupaty. In Chennupaty, hardware logic (*i.e.*, the gates 410, 412, 414 and 420) prevents the decoder 340 from receiving the prefix. It is not the adjustment of a program counter that causes the prefix to be rejected by the decoder 340, as required by claim 1; instead, it is hardware logic that is responsible for preventing the decoder 340 from receiving the prefix. The fact that the decode logic of claim 1 skips the prefix by adjusting the program counter (*i.e.*, adjusting software instead of adjusting hardware (decoders 430 and 340) using hardware signals) is significant at least because it confers many useful benefits. For example, adjusting software instead of hardware entails fewer points of potential failure, increased speed, increased efficiency, decreased space (real estate) requirements, *etc.*

The Examiner asserts, "Further note that if the first byte is a valid prefix, the first byte is ignored by the decoder ... in effect causing the program counter (pointer to the current instruction) to skip the prefix bytes and point to the instruction being decoded" (p. 3 of the Office Action dated September 6, 2006). Appellants provide two comments in response. First, Chennupaty's program counter setup is not explicitly disclosed. The program counter may very well point to the next instruction to be fetched, and if this is the case, then the program counter would not be altered by the decoder 340's rejection of the prefix (since the prefix has already been issued from the instruction buffer 310). Second, what happens to Chennupaty's program counter as a result of the decoder 340's rejection of the prefix is irrelevant. Claim 1 states that the "program counter [skips] the prefix," thereby ***causing*** the decoder to reject the prefix. In Chennupaty, the cause-effect relationship is reversed from what is required by claim 1: adjustment of the program counter, if it happens at all, would be an ***effect*** of the decoder 340's rejection of the prefix. Narayan, Google, JVM and Nazomi all fail to satisfy the deficiencies of Chennupaty.

In the Final Office Action (p. 3), the Examiner references col. 6, ll. 38-53 of

Chennupaty as disclosing the limitation referenced above. However, this portion of Chennupaty appears to merely reiterate the teachings of Chennupaty described above, albeit in a flowchart format.

In summary, all combinations of the cited references fail to teach precluding a prefix from being received by a decoder by causing a program counter to skip the prefix. Specifically, Chennupaty discloses that the decoder 340 is prevented from receiving the prefix due to the hardware gates 410, 412, 414 and 420 (***not*** due to the adjustment of a program counter), and Narayan, Google, JVM and Nazomi all fail to satisfy the deficiencies of Chennupaty.

Based on the foregoing, Appellant respectfully submits that the rejections of the claims in this grouping be reversed, and the claims set for issue.

2. Rejections in View of Chennupaty, Narayan and Google

Claims 3-4, 13, 15-16 and 27 stand rejected under 35 U.S.C. § 103(a) as allegedly obvious in view of Chennupaty, Narayan and Google. Each of these claims is allowable for at least the same reasons as those described in Section VII(B)(1) above, and further because Google fails to satisfy the deficiencies of the combination of Chennupaty and Narayan. Thus, Appellants respectfully submit that the rejections of the claims in this grouping be reversed, and the claims set for issue.

3. Rejections in View of Chennupaty, Narayan and JVM

Claims 5-6, 14 and 26 stand rejected under 35 U.S.C. § 103(a) as allegedly obvious in view of Chennupaty, Narayan and JVM. Each of these claims is allowable for at least the same reasons as those described in Section VII(B)(1) above, and further because JVM fails to satisfy the deficiencies of the combination of Chennupaty and Narayan. Thus, Appellants respectfully submit that the rejections of the claims in this grouping be reversed, and the claims set for issue.

4. Rejections in View of Chennupaty, Narayan and Nazomi

Claims 17-20, 23 and 24 stand rejected under 35 U.S.C. § 103(a) as allegedly obvious in view of Chennupaty, Narayan and Nazomi. Claim 17 is representative of this group of claims. The grouping should not be construed to mean the patentability of any of the claims may be determined in later actions (*e.g.*, actions before a court) based on the groupings. Rather, the presumption of 35 USC § 282 shall apply to each of these claims individually.

Claim 17 requires "wherein the decode logic decodes a current instruction concurrently with the pre-decode logic determining if a subsequent instruction comprises a prefix in which case a program counter skips the prefix thereby precluding the decode logic from receiving the prefix and changes the decode logic operation during the decoding of the subsequent instruction." The Examiner maps this limitation to Chennupaty. However, as described above in Section VII(B)(1), the combination of Chennupaty and Narayan fails to teach or suggest such a limitation. Nazomi also fails to teach this limitation. Because Nazomi fails to satisfy the deficiencies of the combination of Chennupaty and Narayan, the Examiner erred in rejecting claim 17 in view of Chennupaty, Narayan and Nazomi.

Based on the foregoing, Appellants respectfully submit that the rejections of the claims in this grouping be reversed, and the claims set for issue.

5. Rejections in View of Chennupaty, Narayan, Nazomi and Google

Claim 21 stands rejected under 35 U.S.C. § 103(a) as allegedly obvious in view of Chennupaty, Narayan, Nazomi and Google. Claim 21 is allowable for at least the same reasons as those described in Section VII(B)(4) above, and further because Google fails to satisfy the deficiencies of the combination of Chennupaty, Narayan and Nazomi. Thus, Appellants respectfully submit that the rejection of this claim be reversed, and the claim set for issue.

6. Rejections in View of Chennupaty, Narayan, Nazomi and JVM

Claim 22 stands rejected under 35 U.S.C. § 103(a) as allegedly obvious in view of Chennupaty, Narayan, Nazomi and JVM. Claim 22 is allowable for at least the same reasons as those described in Section VII(B)(4) above, and further because JVM fails to satisfy the deficiencies of the combination of Chennupaty, Narayan and Nazomi. Thus, Appellants respectfully submit that the rejection of this claim be reversed, and the claim set for issue.

C. Conclusion

For the reasons stated above, Appellants respectfully request that the rejections be reversed, and the claims set for issuance. It is believed that no extensions of time or fees are required, beyond those that may otherwise be provided for in documents accompanying this paper. However, in the event that additional extensions of time are necessary to allow consideration of this paper, such extensions are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required (including fees for net addition of claims) are hereby authorized to be charged to Texas Instruments, Inc.'s Deposit Account No. 20-0668.

Respectfully submitted,

/Nick P. Patel/

Nick P. Patel
PTO Reg. No. 57,365
Conley Rose, P.C.
(713) 238-8000 (Phone)
(713) 238-8008 (Fax)
AGENT FOR APPELLANTS

VIII. CLAIMS APPENDIX

1. (Previously presented) A processor, comprising:

instruction storage in which instructions are stored;

fetch logic coupled to the instruction storage to fetch instructions from the instruction storage;

decode logic coupled to the fetch logic to decode instructions fetched by the fetch logic; and

pre-decode logic associated with the decode logic;

wherein at least some of the instructions comprise a prefix, and in parallel with the decode logic decoding a current instruction, the pre-decode logic determines whether a subsequent instruction comprise a prefix, in which case the decode logic causes a program counter to skip the prefix thereby precluding the decode logic from receiving the prefix and changes behavior of the decode logic during decoding of the subsequent instruction.

2. (Original) The processor of claim 1, wherein at least some instructions comprise at least one Bytecode.

3. (Original) The processor of claim 1, wherein the prefix comprises a Java impdep instruction.

4. (Original) The processor of claim 3, wherein when detecting the Java impdep instruction, the subsequent instruction belongs to a different instruction set than the current instruction.

5. (Original) The processor of claim 1, wherein the prefix comprises a Java wide instruction.

6. (Original) The processor of claim 1, wherein when detecting the Java wide instruction changes format of the subsequent instruction.

7. (Original) The processor of claim 1, wherein in parallel with the decode logic decoding the current instruction, the pre-decode logic examines a predetermined number of subsequent bytes.

8. (Original) The processor of claim 7, wherein the predetermined number is at least 5.

9. (Previously presented) A method of decoding variable length instructions, comprising:
decoding a current instruction according to a first behavior;
while decoding the current instruction, pre-decoding a subsequent instruction to determine if the subsequent instruction includes a predetermined prefix; and
if the subsequent instruction includes the predetermined prefix, causing a program counter to skip the predetermined prefix to thereby preclude decode logic from receiving the prefix and changing the decoding of the subsequent instruction according to a second behavior.

10. (Original)The method of claim 9, wherein pre-decoding includes examining a predetermined number of bytes following the current instruction.

11. (Original)The method of claim 10, wherein the predetermined number is at least 5.

12. (Original)The method of claim 10, wherein pre-decoding further includes comparing each of the predetermined number of bytes to prefix value.

13. (Original)The method of claim 12, wherein the prefix value is equal to a Java impdep instruction.

14. (Original)The method of claim 12, wherein the prefix value is equal to a Java wide instruction.

15. (Original)The method of claim 9, wherein if the a Java wide prefix is detected, the first and second behaviors comprise a first mode for decoding instructions of a first format and a second mode for decoding instructions of a second format.

16. (Original)The method of claim 9, wherein if a Java impdep prefix is detected, the first and second behaviors comprise a first mode for decoding instructions of a first instruction set and a second mode for decoding instructions of a second instruction.

17. (Previously presented) A system, comprising:
a main processor unit; and
a co-processor unit coupled to the main processor unit, the co-processor unit comprising:
decode logic; and
pre-decode logic associated with the decode logic;
wherein the decode logic decodes a current instruction concurrently with the pre-decode logic determining if a subsequent instruction comprises a prefix in which case a program counter skips the prefix thereby precluding the decode logic from receiving the prefix and changes the decode logic operation during the decoding of the subsequent instruction.

18. (Original)The system of claim 17, wherein concurrently with the decode logic decoding the current instruction, the pre-decode logic examines a predetermined number of subsequent bytes.

19. (Original)The system of claim 18, wherein the predetermined number is at least 5.

20. (Original)The system of claim 18, wherein the predetermined number of subsequent bytes is compared to a prefix value.

21. (Original)The system of claim 20, wherein the prefix value is equal to a Java wide instruction .

22. (Original)The system of claim 20, wherein the prefix value is equal to a Java impdep prefix.

23. (Original) The system of claim 17, wherein the instructions are of variable length.

24. (Original)The system of claim 17, wherein the system comprises a cellular telephone.

25. (Previously presented) A programmable device, comprising:

a register storing a location of a current instruction;

a decode logic; and

a pre-decode logic coupled to the decode logic, wherein in parallel, the decode logic decodes the current instruction and the pre-decode logic determines if a subsequent instruction includes a prefix, and wherein if the subsequent instruction comprises the prefix, the register skips the prefix of the subsequent instruction thereby precluding the decode logic from receiving the prefix and changes behavior of the decode logic for decoding of the subsequent instruction.

26. (Original)The programmable device of claim 25, wherein the prefix is a Java wide instruction.

27. (Original)The programmable device of claim 25, wherein the prefix is a Java impdep instruction.

28. (Original)The programmable device of claim 25 wherein the current instruction and the subsequent instruction each comprises at least one Bytecode.

29. (Original)The programmable device of claim 25, wherein the pre-decode logic further determines a predetermined number of subsequent bytes.

30. (Original)The programmable device of claim 29, wherein the predetermined number is at least 5.

31. (Original)The programmable device of claim 25, wherein the register is a program counter.

32.-34. (Canceled).

IX. EVIDENCE APPENDIX

None.

X. RELATED PROCEEDINGS APPENDIX

None.